Creating an isolated Python environment

ابتدا یک پوشه برای پروژه درست کنید سپس در آن پوشه محیط cmd را بالا آورید و بعد در cmd دستور زیر را وارد نمایید تا یک محیط مجازی برای پروژه شما ایجاد کند.

python -m venv .env

ادیتور vscode را باز کرده در قسمت extensions در قسمت پایین گزینه پایتون روی setting کلیک کرده و گزینه extension setting را انتخاب کرده و در فرمی که باز می شود آدرس مفسر پایتون را به آدرس محیط مجازی نصب شده تغییر دهید. بعد، از منوی فایل گزینه open folder را انتخاب کرده و فولدر پروژه با انتخاب کنید.

در قسمت ترمینال اگر محیط مجازی برایتان فعال نشده دستور زیر را تایپ کنید.

.env\Scripts\activate

اگر محیط مجازی شما فعال شود قبل از آدرس فولدر در ترمینال، عبارت (env) ظاهر می شود. در این مرحله با د ستور زیر جنگو را نصب کنید.

pip install Django

جهت اطمینان از نصب و نمایش ورژن جنگو در ترمینال دستور زیر را تاب کنید.

```
django-admin --version
```

یا با نوشتن واژه py در ترمینال و فشرن دکمه اینتر وارد محیط shell شده و کد های زیر را بنویسید

import Django

django.get_version()

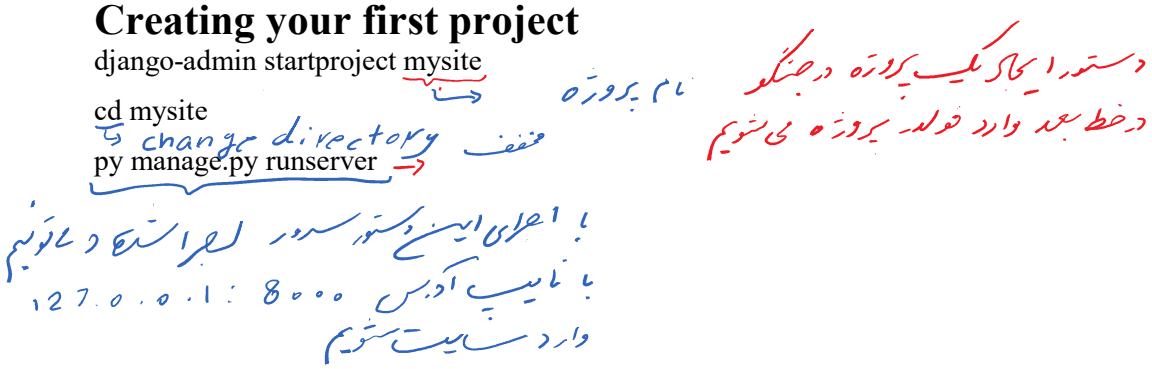
در صورتی که با خطایی مواجه نشدین تبریک می گم شما موفق شدین جنگو را نصب کنید. حال باید اولین پروژه جنگویی خود را ایجاد کنید

Creating your first project

django-admin startproject mysite

cd mysite

py manage.py runserver



با دستور زیر دیتابیس پروژه و جداول پیشفرض جنگو در دیتابیس ایجاد می شوند.

py manage.py migrate

می توانید با باز کردن فایل setting.py در داخل فولدر پروژه مقادیر زیر را به مقدار گفته شده تغییر دهید.

LANGUAGE_CODE = "fa-ir"

TIME_ZONE = 'Asia/Tehran'

Projects and applications

هر پرژه ای ممکن است از چندین تا اپلیکیشن درست شده باشد. اپلیکیشن ها قابلیت استفاده مجدد دارند و سرعت ما را در توسعه پروژه های بعدی بیشتر می کنند. برای ایجاد اپلیکیشن از دستور زیر استفاده کنید.

Creating an application

python manage.py startapp blog

نام اپلیکیشن ما در اینجا blog است

Designing the blog data schema

Model.py

```
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User
# Create your models here.
class Post(models.Model):
    STATUS_CHOICES=(
        ('draft','پیشنویس'),
        ('published','منتشر شده')
    )
    title=models.CharField(max_length=250)
    slug=models.SlugField(max_length=250,unique_for_date='publish')

author=models.ForeignKey(User,on_delete=models.CASCADE,related_name='blog
_posts')
    body=models.TextField()
    publish=models.DateTimeField(default=timezone.now)
    created=models.DateTimeField(auto_now_add=timezone.now)
    updated=models.DateTimeField(auto_now=timezone.now)

status=models.CharField(max_length=10,choices=STATUS_CHOICES,default='dra
ft')
    class Meta:
        ordering=('-publish',)
    def __str__(self):
        return self.title
```

در فایل model.py اپلیکیشن blog کدهای زیر را اضافه میکنیم و در اینجا ویژگی های کلاس پست را ثبت میکنیم

slug یک رشته کوتاه است که میتوانیم از آن در آدرس های url هر پست استفاده کنیم

وقتی از پست نویسنده میخواهیم پست های ایشان دسترسی پیدا کنیم یا چنان پست هایی دسترسی پیدا کنیم

یعنی نویسنده یک پست است و هر نویسنده میتواند چندین پست داشته باشد

مقدار پیش فرض

در حین ایجاد یک پست بطور خودکار زمان فعلی ثبت میشود

Activating the application

```
INSTALLED_APPS = [
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
```

```
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'blog.apps.BlogConfig',
]
```

Creating and applying migrations

python manage.py makemigrations blog

python manage.py migrate

super user for work in admin panel Create

python manage.py createsuperuser

ثبت مدل پست در ادمین پنل

```
from django.contrib import admin
from .models import Post
# Register your models here.
admin.site.register(Post)
```

تغییر شیوه نمایش مدل در ادمین پنل

```
@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
  list_display = ("title", "slug", "author", "publish", "status")
  list_filter = ("publish", "status")
  search_fields = ("title", "body")
  prepopulated_fields = {"slug": ("title",)}

  class Meta:
    ordering = ("status", "-publish")
    verbose_name = "پست"
    verbose_name_plural = "پست ها"
```

فارسی سازی نام اپ در فایل app.py

```
class BlogConfig(AppConfig):
    name = 'blog'
    verbose_name='بلاگ'
```

Working with QuerySets and managers

```
python manage.py shell
>>> from django.contrib.auth.models import User
>>> from blog.models import Post
>>> user = User.objects.get(username='admin')
>>> post = Post(title='Another post',slug='another-post', body='Post body.', author=user)
>>> post.save()
>>> post.title = 'New title'
>>> post.save()
```

Retrieving objects

```
Post.objects.get()
all_posts = Post.objects.all()
```

Using the filter() method

```
Post.objects.filter(publish__year=2020)
Post.objects.filter(publish__year=2020, author__username='admin')
Post.objects.filter(publish__year=2020).filter(author__username='admin')
Post.objects.filter(publish__year=2020) .exclude(title__startswith='Why')
Post.objects.order_by('title')
Post.objects.order_by('-title')
```

Deleting objects

```
>>> post = Post.objects.get(id=1)
>>> post.delete()
```

Creating model managers

```
class PublishedManager(models.Manager):
    def get_queryset(self):
        return super().get_queryset().filter(status='published')
```

هر کلاسی یک منیجر پیشفرض بنام objects دارد که ما هم می توانیم منیجر خاص خودمون را طبق الگوی کد بالا بسازیم. لازم است که منیجر ساخته شده را مثل کد زیر به مدل نیز اضافه کنیم

```
objects=models.Manager()
published=PublishedManager()
```

تست در محیط shell

```
>>> from blog.models import Post
>>> Post.published.filter(title__startswith='Who')
```

تنظیمات آپلود عکس

اول: اضافه کردن این فیلد به مدل

```
thumbnail = models.ImageField(upload_to = 'images', null = True, blank = True)
```

برای استفاده از فیلد ImageField حتما باید کتابخانه pillow نصب شده باشد و اینجا لازم است که با دستور pip install pillow در محیط venv کتابخانه pillow را نصب نمایید

دوم اضافه کردن این دو خط در فایل settings.py

```
MEDIA_URL = "media/"  # new
```

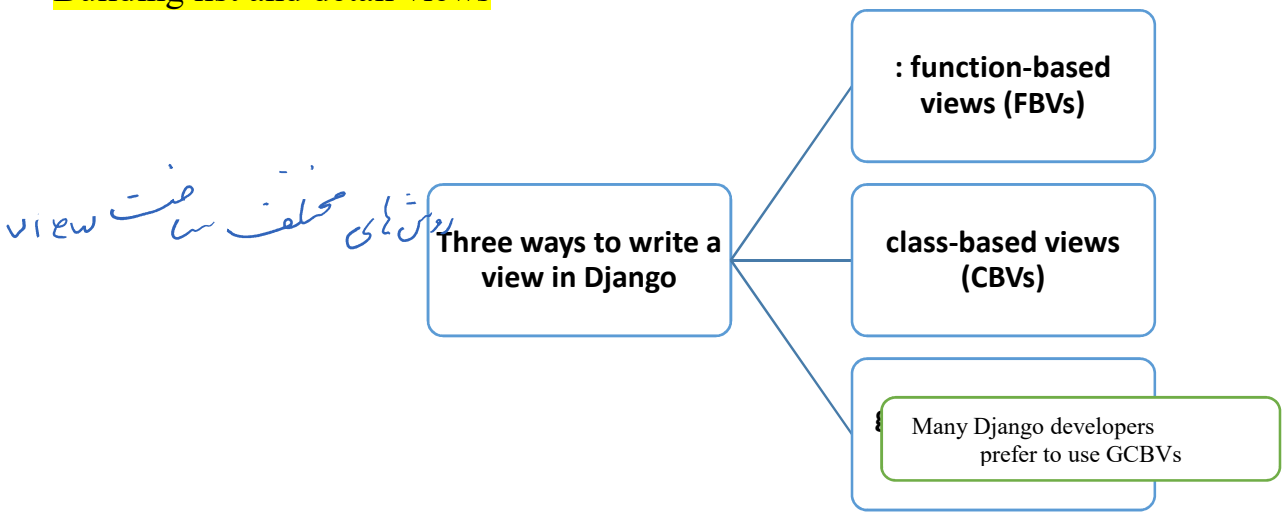
```
MEDIA_ROOT = BASE_DIR / "media"  # new
```

سوم اگر در حالت debug هستید این خط کد ها را به فایل urls.py اصلی پروژه اضافه کنید.

```
from django.conf import settings
from django.conf.urls.static import static

urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Building list and detail views



ویو در جنگو تابعی است که یک request دریافت کرده و یک response مطلوب برمی گرداند

```
Views.py      فایل
from django.shortcuts import render
```

```
from .models import Post
# Create your views here.
def post_list(request):
    posts=Post.published.all()
    return render(request,'blog/post/list.html',{'posts':posts})
```

```
def post_list(request):
    posts = Post.objects.all()
    #posts = Post.published.all()
    #posts = Post.objects.filter(status = 'published').order_by('-published')
    return render(request, "blog/post/list.html", {"posts": posts})
```

Adding URL patterns for your views

Create urls.py

```
from django.urls import path
from . import views

app_name = 'blog'
urlpatterns = [
    path('',views.post_list,name='post_list'),
    path('<int:year>/<int:month>/<int:day>/<slug:post>/',views.post_detail,name='post_detail'),
]
```

Add this url to project urls

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
   path('blog/',include('blog.urls',namespace='blog')),
]
```

Canonical URLs for models

```
    def get_absolute_url(self):
        return reverse('blog:post_detail',
                       args=[self.publish.year,
                             self.publish.month,
                             self.publish.day,
                             self.slug])
```

Creating templates for your views

```
templates/
    blog/
        base.html
        post/
            list.html
            detail.html
```

base.html

```
{% load static %}
<!DOCTYPE html>
<html>
    <head>
        <title>{% block title %}{% endblock %}</title>
        <link href="{% static "css/blog.css" %}" rel="stylesheet">
    </head>
    <body>
        <div id="content">
            {% block content %}
            {% endblock %}
        </div>
        <div id="sidebar">
            <h2>My blog</h2>
            <p>This is my blog.</p>
        </div>
    </body>
</html>
```

list.html

```
{% extends "blog/base.html" %}
{% block title %}My Blog{% endblock %}
{% block content %}
    <h1>My Blog</h1>
    {% for post in posts %}
{% if post.thumbnail %}
<img src = '{{post.thumbnail.url}}' alt = '{{post.title}}'>
{% endif %}

        <h2>
            <a href="{{ post.get_absolute_url }}">
             {{ post.title }}
            </a>
        </h2>
        <p class="date">
             Published {{ post.publish }} by {{ post.author }}
        </p>
        {{ post.body|truncatewords:30|linebreaks }}
    {% endfor %}
{% endblock %}
```

```
href = "{% url 'blog:detail_post' post.slug %}"
```

استفاده از یک قالب آماده

1. پیدا کردن قالب مورد نظر
2. کپی کردن فایل های html داخل فولدر template
3. برای فایل های static ایجاد فولدر static/blog در اپ مورد نظر و کپی کردن فایل های مورد نظر در آن فولدر
4. استفاده از تگ {٪ load static ٪} در تمپلیت و اصلاح آدرس های فایل های استاتیک

استفاده از قابلیت extends

معمولا تمام صفحه ها از هدر و فوتر یکسانی برخوردارند در اینجا یک صفحه ای را به عنوان base می سازیم و block هایی داخل آن قرار می دهیم بعد در هر صفحه ای که لازم باشه از صفحه base اکستند extend می کنیم

تمرین ساخت بلوک main و title

استفاده از قابلیت فیلتر truncate در صفحه index.html و فیلتر linebreak در صفحه detail.html

```
index.html
<h4><a href = "{% url 'blog:detail_post' post.slug %}" title="">{{post.title}}</a></h4>
<p>{{post.body|truncatewords:30}}
        <a href = "{% url 'blog:detail_post' post.slug %}" title="">بیشتر</a>
</p>
```

```
detail.html
<p>{{post.body|linebreaks}}</p>
```

تغییر view برای نمایش خطای 404

```
from django.shortcuts import render, get_object_or_404
def detail_post(request, slug):
    # post = Post.objects.get(slug=slug)
    post = get_object_or_404(Post, slug=slug, status="published")
    return render(request, "blog/post/detail.html", {"post": post})
```

شمسی سازی تاریخ

1- ایجاد فولدر extensions در دایرکتوری پروژه و ساخت فایل خالی __init__.py داخل آن و اضافه کردن "extensions", به فایل setting.py در قسمت INSTALLED_APPS بصورت زیر

```
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    # new
    "blog.apps.BlogConfig",
    "extensions",
]
```

2- مراجعه به آدرس https://raw.githubusercontent.com/mjnaderi/Jalali.py/master/jalali.py و دانلود فایل jalali.py داخل فولدر extensions

3- ساخت فایل utils.py داخل فولدر extensions و نوشتن کدهای زیر داخل آن

```
from . import jalali
from django.utils import timezone
def jalali_convertor(time):
    time = timezone.localtime(time)
    jalali_month = [
       "فروردین","اردیبهشت", "خرداد","تیر", "مرداد","شهریور","مهر","آبان","آذر","دی","بهمن","اسفند",
    ]
    time_to_str = f"{time.year},{time.month},{time.day}"
    time_to_tuple = jalali.Gregorian(time_to_str).persian_tuple()
    time_to_list = list(time_to_tuple)
    for index, month in enumerate(jalali_month):
        if time_to_list[1] == index + 1:
           time_to_list[1] = month
    output = f"{time_to_list[2]} {time_to_list[1]} {time_to_list[0]}, ساعت{time.hour}:{time.minute}"
    return output
```

4- در در نهایت در فایل model.py تابع زیر را اضافه کنید:

```
from extensions import utils
def jalali_publish(self):
        return utils.jalali_convertor(self.publish)
```

5- و از این به بعد هر جا به تاریخ انتشار پست نیاز داشته باشیم بجای ویژگی publish از این تابع استفاده خواهیم کرد.

6- برای فارسی سازی صفحات html می توانید به این لینک مراجعه کنید: https://adminesite.com/html-rtl-/webpage

7- یا

https://mrcode.ir/%D8%A2%D9%85%D9%88%D8%B2%D8%B4-%D8%B1%D8%A7%D8%B3%D8%AA%DA%86%DB%8C%D9%86-%DA%A9%D8%B1%D8%AF%D9%86-%D9%82%D8%A7%D9%84%D8%A8-%D9%88%D8%B1%D8%AF%D9%BE%D8%B1%D8%B3

8- برای شمسی سازی در ادمین پنل نیز تغییر زیر را اعمال می کنیم

```
from django.contrib import admin
from .models import Post
# Register your models here.
# admin.site.register(Post)
@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
```

```
    list_display = ("title", "slug", "author", "jalali_publish", "status")
    list_filter = ("publish", "status")
    search_fields = ("title", "body")
    prepopulated_fields = {"slug": ("title",)}
    class Meta:
```

```
        ordering = ("status", "-publish")
        verbose_name = "پست"
        verbose_name_plural = "پست ها"
```

9- و بعد در مدل در انتهای کلاس پست این خط را اضافه می کنیم

```
jalali_publish.short_description = "تاریخ انتشار"
```

فارسی سازی سازی اعداد

1. اضافه کردن تابع زیر به فایل utils.py

```
def english_number_to_persian(date):
    numbers = {
        "0": "۰",
        "1": "۱",
        "2": "۲",
        "3": "۳",
        "4": "۴",
        "5": "۵",
        "6": "۶",
        "7": "۷",
        "8": "۸",
        "9": "۹",
    }
    for k, v in numbers.items():
        date = date.replace(k, v)
    return date
```

2. تغییر خط آخر تابع jalali_converter(time) به:

```
return english_number_to_persian(output)
```

ساخت مدل Category

1- تغییر مدل بصورت زیر

```
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User
from extensions import utils
# Ceate your models here.
class PublishedManager(models.Manager):
    def get_queryset(self):
        return super().get_queryset().filter(status="published")

class Category(models.Model):
    title = models.CharField(max_length=250, verbose_name="عنوان دسته بندی")
    slug = models.SlugField(max_length=250, verbose_name="آدرس دسته بندی")
    status = models.BooleanField(default=True, verbose_name="آیا نمایش داده شود؟")
    position = models.IntegerField(verbose_name="موقعیت نمایش")

    class Meta:
        verbose_name = "دسته بندی"
        verbose_name_plural = "دسته بندی ها"
        ordering = ("-position",)

    def __str__(self):
        return self.title

class Post(models.Model):
    STATUS_CHOICES = (("draft", "پیشویس"), ("published", "منتشر شده"))
    title = models.CharField(max_length=250, verbose_name="عنوان")
    slug = models.SlugField(
        max_length=250, unique_for_date="publish", verbose_name="آدرس"
    )
    category = models.ManyToManyField(Category, verbose_name="دسته بندی")
    author = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        related_name="blog_posts",
        verbose_name="نویسنده",
    )
    thumbnail = models.ImageField(
        upload_to="images", null=True, blank=True, verbose_name="تصویر"
    )
    body = models.TextField(verbose_name="محتوی")
```

```
    publish = models.DateTimeField(default=timezone.now(), verbose_name="تاریخ انتشار")
    created = models.DateTimeField(
        auto_now_add=timezone.now(), verbose_name="تاریخ ایجاد"
    )
    updated = models.DateTimeField(
        auto_now=timezone.now(), verbose_name="تاریخ بروز رسانی"
    )
    status = models.CharField(
        max_length=10,
        choices=STATUS_CHOICES,
        default="draft",
        verbose_name="وضعیت انتشار",
    )

    class Meta:
        verbose_name = "مقاله"
        verbose_name_plural = "مقالات"
        ordering = ("-publish",)

    def __str__(self):
        return self.title

    def jalali_publish(self):
        return utils.jalali_converter(self.publish)

    jalali_publish.short_description = "تاریخ انتشار"

    objects = models.Manager()
    published = PublishedManager()
```

2- نوشتن دستورهای makemigration, migrate در ترمینال

3- اضافه کردن این کد در admin.py

```
@admin.register(Category)
class CategoryAdmin(admin.ModelAdmin):
    list_display = ("position", "title", "slug", "status")
    list_filter = ["status"]
    search_fields = ("title",)
    prepopulated_fields = {"slug": ("title",)}
```

نمایش دسته بندی در نوبار

1- ارسال دسته بندی ها به صفحه اصلی از طریق اصلاح view

```
def post_list(request):
    # posts = Post.objects.all()
    posts = Post.published.all()
    categories = Category.objects.filter(status=True)
    # posts = Post.objects.filter(status = 'published').order_by('-published')
    return render(
        request, "blog/post/index.html", {"posts": posts, "categories": categories}
    )
```

-2 اصلاح تمپلیت base.html برای نمایش دسته بندی ها با استفاده از حلقه for
-3

```
<ul class="navbar-nav mr-auto">
      <li class="nav-item">
      <a class="nav-link" href="{% url 'blog:post_list' %}">خانه</a>
        </li>
        {% for cat in categories %}
            <li class="nav-item">
                  <a class="nav-link" href="#">{{cat.title}}</a>
            </li>
        {% endfor %}
```

-4 و اصلاح تمپلیت index.html برای نمایش دسته بندی ها هر پست زیر آن

```
 <P>
    {% for cat in post.category.all %}
        <small class="firstsmall" ><a href="#" title="">#{{cat.title}}</a></small>
    {% endfor %}
</p>
```

ایجاد template tag اختصاصی

1. ایجاد یک فولدر به نام templatetags در اپ مورد نظر
2. ایجاد فایل __init__.py در پوشه templatetags
3. ایجاد یک فایل پایتونی و نوشتن template tag داخل آن

```
from django import template
from ..models import Category

register = template.Library()

@register.inclusion_tag("blog/post/partials/category.html")
```

```
def category_navbar():
    return {"categories": Category.objects.filter(status=True)}
```

4. استفاده از این template tag در هر صفحه ای لازم باشه مطابق راهنمای خود جنگو

ایجاد صفحه دسته بندی (نمایش مقالات یک دسته بندی خاص در یک صفحه)

1. ایجاد یک view در فایل view.py

```
def category_list(request, slug):
    category = get_object_or_404(Category, slug=slug, status=True)
  return render(request, "blog/post/category_list.html", {"category": category})
```

2. ساخت تمپلیت category_list.html
3. ایجاد url

```
path("category/<slug:slug>", views.category_list, name="category_list"),
```

4. اصلاح آدرس ها و لینک های دسته بندی ها
5. ایجاد یک تابع در مدل post برای فیلتر کردن دسته بندی های قابل نمایش

```
    def viewable_category(self):
        return self.category.filter(status=True)
```

6. اصلاح حلقه های مربوط که دسته بندی در تمپلیت ها
7. و اصلاح admin.py جهت نمایش ندادن دسته بندی های تایید نشده

```
    def all_category(self, obj):
        return "، ".join([category.title for category in
obj.viewable_category()])
```

اضافه کردن pagination

https://simpleisbetterthancomplex.com/tutorial/2016/08/03/how-to-paginate-with-django.html

1. اضافه کردن این مسیر به urls.py

```
path("page/<int:number>", views.post_list, name="post_list"),
```

2. اعمال تغییرات زیر در صفحه view.py

```
from django.core.paginator import Paginator

# Create your views here.
```

```
def post_list(request, number=1):
    # posts = Post.objects.all()
    posts_list = Post.published.all()
    paginator = Paginator(posts_list, 5)
    posts = paginator.get_page(number)
    # categories = Category.objects.filter(status=True)
    # posts = Post.objects.filter(status = 'published').order_by('-published')
    return render(
        request,
        "blog/post/index.html",
        {
            "posts": posts,
        },  # "categories": categories}
    )
```

3. اصلاح قسمت صفحه بندی در تمپلیت

```
{% if posts.has_other_pages %}
   <nav aria-label="Page navigation">
       <ul class="pagination justify-content-start">
           {% if posts.has_previous %}
             <li><a href="{% url 'blog:post_list' posts.previous_page_number
%}">&laquo;</a></li>
           {% endif %}
   {% for i in posts.paginator.page_range %}
      {% if posts.number == i %}
       <li class="page-item"><a class="page-link active" href="{% url
'blog:post_list' i %}">{{i}}</a></li>
     {% else %}
       <li class="page-item"><a class="page-link" href="{% url 'blog:post_list'
i %}">{{i}}</a></li>
     {% endif %}
 {% endfor %}


     {% if posts.has_next %}
           <li><a href="{% url 'blog:post_list' posts.next_page_number
%}">&raquo;</a></li>
     {% endif %}
</ul>
</nav>
{% endif %}
```

اضافه کردن pagination به صفحه دسته بندی

1. اصلاح view.py

```
def category_list(request, slug, number=1):
    category = get_object_or_404(Category, slug=slug, status=True)
    posts_list = category.posts.filter(status="published")
    paginator = Paginator(posts_list, 2)
    posts = paginator.get_page(number)
    return render(
        request, "blog/post/category_list.html", {"category": category, "posts": posts}
    )
```

2. اصلاح url

```
app_name = "blog"

urlpatterns = [
    path("", views.post_list, name="post_list"),
    path("page/<int:number>", views.post_list, name="post_list"),
    re_path(r"post/(?P<slug>[-\w]+)", views.detail_post, name="detail_post"),
    path("category/<slug:slug>", views.category_list, name="category_list"),
    path(
        "category/<slug:slug>/page/<int:number>",
        views.category_list,
        name="category_list",
    ),
]
```

3. اصاح تمپلیت category list

```
{% if posts.has_other_pages %}
   <nav aria-label="Page navigation">
     <ul class="pagination justify-content-start">
       {% if posts.has_previous %}
         <li><a href="{% url 'blog:category_list' category.slug posts.previous_page_number %}">&laquo;</a></li>
       {% endif %}
       { for i in posts.paginator.page_range %}
```

```
            {% if posts.number == i %}
```

```
            <li class="page-item"><a class="page-link active" href="{% url
'blog:category_list' category.slug i %}">{{i}}</a></li>
        {% else %}
            <li class="page-item"><a class="page-link" href="{% url
'blog:category_list' category.slug i %}">{{i}}</a></li>
        {% endif %}
    {% endfor %}


    {% if posts.has_next %}
        <li><a href="{% url 'blog:category_list' category.slug
posts.next_page_number %}">&raquo;</a></li>
    {% endif %}
</ul>
</nav>
{% endif %}
```

فارسی کردن slug

1. ایمپورت کردن re_path در فایل urls.py
2. تغییر آدرس صفحه detail به

```
re_path(r"post/(?P<slug>[-\w]+)", views.detail_post, name="detail_post")
```

3. اضافه کردن آرگومان allow_unicode=True, به ویژگی slug در مدل post
4. در admin.py حتما این ویژگی ست شده باشد

```
prepopulated_fields = {"slug": ("title",)}
```

دسته بندی های تو در تو

1. اضافه کردن فیلد زیر در مدل category

```
parent = models.ForeignKey(
        "self",
        default=None,
        null=True,
        blank=True,
        related_name="children",
        verbose_name="زیر دسته بندی",
        on_delete=models.SET_NULL,)
```

دقت شود که self داخل quotation mark نوشته شود

2. تغییر oredering در کلاس meta

```
ordering = (
            "parent__id",
            "position",
        )
```

3. اصلاح admin.py

```
list_display = ("position", "title", "slug", "parent", "status")
```

4. مرحله آخر migrate

اصلاح نمایش تو در تو دسته بندی ها در تمپلیت

1. تغییر در فایل base.html و برداشتن کل تگ nav قراردادن آن تگ در فایل category.html در قسمت partial

```
<nav class="navbar navbar-toggleable-md navbar-inverse fixed-top bg-inverse">
    <button class="navbar-toggler navbar-toggler-right" type="button" data-
toggle="collapse" data-target="#navbarCollapse" aria-controls="navbarCollapse" aria-
expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <a class="navbar-brand" href="tech-index.html"><img src="images/version/tech-logo.png"
alt=""></a>
    <div class="collapse navbar-collapse" id="navbarCollapse">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item">
                <a class="nav-link" href="{% url 'blog:post_list' %}">خانه</a>
            </li>
            {% for cat in categories %}
            {% if cat.parent == None %}
                <li class="nav-item">
                    <a class="nav-link" href="{% url 'blog:category_list' cat.slug
%}">{{cat.title}}</a>
                    {% if cat.children.all %}
                        {% include 'blog\post\partials\nested_category.html' %}
                    {% endif %}

                </li>
            {% endif %}
        {% endfor %}
            <li class="nav-item">
                <a class="nav-link" href="tech-contact.html">ارتباط با ما</a>
            </li>
        </ul>
        <ul class="navbar-nav mr-2">
            <li class="nav-item">
                <a class="nav-link" href="#"><i class="fa fa-rss"></i></a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#"><i class="fa fa-android"></i></a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#"><i class="fa fa-apple"></i></a>
            </li>
        </ul>
    </div>
</nav>
```

2. ایجاد یک فایل به نام nested_category.html در فولدر partial و نوشتن کد زیر

```
<ul class="navbar-nav mr-auto">
    {% for cat in cat.children.all %}
    <li class="nav-item">
        <a class="nav-link" href="{% url 'blog:category_list' cat.slug
%}">{{cat.title}}</a>
        {% if cat.children.all %}
            {% include 'blog\post\partials\nested_category.html' %}
        {% endif %}
    </li>
    {% endfor %}
</ul>
```

استفاده از (ListView) class base views

1. ایمپورت from django.views.generic import ListView در فایل views.py
2. کامنت کردن کل تابع post_list و اضافه کردن این کلاس

```
class PostList(ListView):
    # model = Post
    queryset = Post.published.all()
    context_object_name = "posts"
    paginate_by = 3
    template_name = "blog/post/index.html"
```

3. اصلاح url

```
    # path("", views.post_list, name="post_list"),
    # path("page/<int:number>", views.post_list, name="post_list"),
    path("", views.PostList.as_view(), name="post_list"),
    path("page/<int:number>", views.PostList.as_view(), name="post_list"),
```

4. تغییر متغیر posts به page_obj در قسمت pagination مربوط به تمپلیت

استفاده از (DetailView) class base views

1. تغییر views.py

```
class PostDetail(DetailView):
    def get_object(self):
        slug = self.kwargs.get("slug")
        return get_object_or_404(Post, slug=slug, status="published")

    context_object_name = "post"
    template_name = "blog/post/detail.html"
```

2. اصلاح urls.py

ساخت صفحه لیست مقالات یک دسته بندی با استفاده از class base views

1. تغییر views.py

```
class CategoryList(ListView):
    paginate_by = 2
    template_name = "blog/post/category_list.html"
    context_object_name = "posts"

    def get_queryset(self):
        global category
        slug = self.kwargs.get("slug")
        category = get_object_or_404(Category, slug=slug, status=True)
        return category.posts.filter(status="published")

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["category"] = category
        return context
```

2. اصلاح urls و تغییر متغیر posts به page_obj در قسمت pagination مربوط به تمپلیت

اکشن در پنل مدیریت

1. اضافه کردن این کدها در کلاس PostAdmin در فایل Admin.py

```
    def make_publish(self, request, queryset):
        row_updated = queryset.update(status="published")
        if row_updated == 1:
            message_bit = "منتشر شد."
        else:
            message_bit = "منتشر شدند."
        self.message_user(request, f"{row_updated} مقاله{message_bit}")
```

```
    make_publish.short_description = "تغییر به منتشر شده"

    def make_draft(self, request, queryset):
        row_updated = queryset.update(status="draft")
        if row_updated == 1:
            message_bit = "پیشنویس شد."
        else:
            message_bit = "پیشنویس شدند."
```

```
        self.message_user(request, f"{row_updated} مقاله{message_bit}")

    make_draft.short_description = "تغییر به پیشنویس"
```

اصلاح تمپلیت ها و جلوگیری از تکرار کد نویسی

1. تغییر نام index.html به list.html
2. ایجاد دو فایل به نام های post_list.html و category_list.html و قراردادن کد های زیر در آنها
3. Post_list.html

```
{% extends 'blog\post\list.html' %}
{% block title %}لیست اخبار{% endblock title %}
{% block previos_page %}
    {% url 'blog:post_list' page_obj.previous_page_number %}
{% endblock previos_page %}

{% block page_link %}
    {% url 'blog:post_list' i %}
{% endblock page_link %}
{% block active_page_link %}
    {% url 'blog:post_list' i %}
{% endblock active_page_link %}

{% block next_page %}
    {% url 'blog:post_list' page_obj.next_page_number %}
{% endblock next_page %}
```

4. Category_list.html

```
{% extends 'blog\post\list.html' %}
```

```
{% block title %} دسته بندی{{category.title}} {% endblock title %}

{% block previos_page %}
    {% url 'blog:category_list' category.slug page_obj.previous_page_number %}
{% endblock previos_page %}

{% block page_link %}
```

```
    {% url 'blog:category_list' category.slug i %}
{% endblock page_link %}
{% block active_page_link %}
    {% url 'blog:category_list' category.slug i %}
{% endblock active_page_link %}

{% block next_page %}
    {% url 'blog:category_list' category.slug page_obj.next_page_number %}
{% endblock next_page %}
```

5. تغییر و ایجاد بلوک های مناسب در قسمت pagination صفحه list.html

```
{% if page_obj.has_previous %}
     <li><a href="{% block previos_page %}{% endblock previos_page
%}">&laquo;</a></li>
{% endif %}
{% for i in page_obj.paginator.page_range %}
      {% if page_obj.number == i %}
         <li class="page-item"><a class="page-link active" href="{% block
active_page_link %}{% endblock active_page_link %}">{{i}}</a></li>
     {% else %}
         <li class="page-item"><a class="page-link" href="{% block page_link
%}{% endblock page_link %}">{{i}}</a></li>
   {% endif %}
{% endfor %}
{% if page_obj.has_next %}
     <li><a href="{% block next_page %}{% endblock next_page
%}">&raquo;</a></li>
{% endif %}
```

6. تغییر متغییر posts به object_list و post به obj در صفحه list.html
7. حذف متغییر context_object_name در کلاس های CategoryList و PostList در فایل views.py

ایجاد لیست مقالات یک نویسنده خاص

1. ساخت دو تا url

```
    path("author/<slug:username>", views.AuthorList.as_view(),
name="author_list"),
    path(
        "author/<slug:username>/page/<int:number>",
        views.AuthorList.as_view(),
        name="author_list",
```

```
    ),
```

2. ساخت یک کلاس در view

```
class AuthorList(ListView):
    paginate_by = 2
    template_name = "blog/post/author_list.html"
    def get_queryset(self):
        global author
        username = self.kwargs.get("username")
        author = get_object_or_404(User, username=username)
        return author.posts.filter(status="published")

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["author"] = author
        return context
```

3. ساخت تمپلیت author_list.html

```
{% extends 'blog\post\list.html' %}
{% block title %} مقالات{{author.get_full_name}} {% endblock title %}

{% block previos_page %}
    {% url 'blog:author_list' author.username page_obj.previous_page_number %}
{% endblock previos_page %}

{% block page_link %}
    {% url 'blog:author_list' author.username i %}
{% endblock page_link %}
{% block active_page_link %}
    {% url 'blog:author_list' author.username i %}
{% endblock active_page_link %}

{% block next_page %}
    {% url 'blog:author_list' author.username page_obj.next_page_number %}
{% endblock next_page %}
```

4. ایجاد تگ نویسنده در list.html

```
<small><a href="{% url 'blog:author_list' obj.author %}"
title="">{{obj.author.get_full_name}}</a></small
```

کار با صفحه لاگین و ادمین پنل شخصی

1. ایجاد اپلیکیشن اکانت
2. ثبت اپلیکیشن در فایل setting.py
3. کپی فایل urls.py از مسیر env\Lib\site-packages\django\contrib\auth\urls.py به فولدر اپلیکیشن اکانت
4. اضافه کردن url زیر به url های پروژه

```
path("account/", include("account.urls", namespace="account")),
```

5. خارج کردن پوشه تمپلیت از اپ بلاگ و قرار دادن در root
6. تغییر .dirs در بخش templates در فایل setting.py

```
"DIRS": [BASE_DIR / 'templates'],
```

7. ایجاد فولدر registration در فولدر templates و ایجاد فایل login.html داخل آن
8. قرار دادن کدهای زیر در login.html

```
<form method = 'post'>
    {% csrf_token %}
    {{form.as_p}}
    <button type = 'submit'> ورود</button>
</form>
```

9. تعریف متغییرهای زیر در setting.py

```
LOGIN_REDIRECT_URL = "account:home"
LOGIN_URL = "account:login"
```

10. ساخت home.html در فولدر registration
11. ساخت view بصورت زیر

```
from django.shortcuts import render
from django.contrib.auth.decorators import login_required
# Create your views here.
@login_required()
def home(request):
    return render(request, "registration/home.html")
```

12. پیدا کردن و تنظیم یک تمپلیت مناسب برای login

نمایش خطاهای لاگین

1. تغییر login.html به صورت زیر

```
{% if form.non_field_errors %}
    {% for error in form.non_field_errors %}
        <p>{{error}}</p>
    {% endfor %}
{% endif %}
<form method = 'post'>
    {% csrf_token %}
    {% if form.username.errors %}
        {% for error in form.username.errors %}
            <p>{{error}}</p>
        {% endfor %}
    {% endif %}
    <p>{{form.username}}</P>
    {% if form.password.errors %}
            {% for error in form.password.errors %}
                <p>{{error}}</p>
            {% endfor %}
    {% endif %}
    <p>{{form.password}}</P>
    <button type = 'submit'> ورود</button>
</form>
```

ادمین پنل شخصی

1. مراجعه به آدرس https://github.com/Hesammousavi/PersianAdminLTE و یا https://afrafile.com/f/1 و دانلود کل فایل ها
2. Extract کردن و حذف تمامی فایل ها بجز index.html و فولدرهای plugins و dist
3. خارج کردن دایرکتوری static از اپ blog به دایرکتوری پروژه
4. کپی کردن فولدرهای plugins و dist به فولدر static/adminlte
5. کپی کردن index.html به templates/ adminlte و تغییر نام به base.html
6. آدرس دهی فایل های static
7. ساخت یک فایل بنام home.html و اکستند از base.html
8. تغییر تمپلیت ویوی home در اپ account به home.html فولدر adminlte
9. ایجاد یک خط کد جدید در setting.py

```
STATICFILES_DIRS = [BASE_DIR / "static",]  # new
```

10. جدا کردن سایدبار

تنظیمات بیشتر ادمین پنل شخصی

1. ایجاد یک بلوک title با مقدار پیشفرض کنترل پنل
2. ایجاد بلوک main و اصلاح فوتر

نمایش لیست مقالات در پنل مدیریت

1. برداشت یک جدول از adminlte اصلی و قراردادن در صفحه home
2. تغییر views.py اپ account به

```
from django.shortcuts import render
from blog.models import Post
from django.contrib.auth.mixins import LoginRequiredMixin
from django.views.generic import ListView
# Create your views here.
class PostList(LoginRequiredMixin, ListView):
    model = Post
    template_name = "adminlte/home.html"
```

3. برداشـــتن تابع all_category از admin.py اپ blog و قرار دادن آن در مدل اپ blog به عنوان یک تابع کلاس Post بصور زیر

```
    def all_category(self):
        return "،".join([category.title for category in
self.viewable_category()])
    all_category.short_description = "دسته بندی"
```

4. قرار دادنPost ها در صفحه home

نمایش لیست مقالات هر نویسنده براساس سطح دسترسی در پنل مدیریت

1. تغییر views.py در اپ account

```
class PostList(LoginRequiredMixin, ListView):
    # model = Post
    template_name = "adminlte/home.html"

    def get_queryset(self):
        if self.request.user.is_superuser:
            return Post.objects.all()
        else:
            return Post.objects.filter(author=self.request.user)
```

ایجاد مقاله جدید از طریق ادمین شخصی

1. ساخت view

```
from django.views.generic import ListView, CreateView
from django.urls import reverse_lazy
class PostCreate(LoginRequiredMixin, CreateView):
```

```
    model = Post
    fields = [
        "title",
        "slug",
        "category",
        "author",
        "thumbnail",
        "body",
        "status",
    ]
    template_name = "adminlte/post_create_update.html"
    success_url = reverse_lazy("account:home")
```

2. ساخت url

```
path("post/create", views.PostCreate.as_view(), name="post_create"),
```

3. ساخت تمپلیت

```
{% extends 'adminlte/base.html' %}
{% block title %}ایجاد مقالات{% endblock title %}
{% block main %}
 <form method="post" enctype="multipart/form-data" >
    {% csrf_token %}
    {{form.as_p}}
    <button type = 'submit' class="btn btn-primary">ارسال</button>
 </form>
{% endblock main %}
```

4. ایجاد لینک "ایجاد مقاله" در sidebar

Crispy

1. نصب کتابخانه مورد نظر pip install django-crispy-forms
2. مراجعه به این آدرس و انجام مراحل ذکر شده

https://pypi.org/project/crispy-bootstrap4/

3. pip freeze > requirements.txt
4. مراجعه به این آدرس و انجام مراحل ذکر شده

https://simpleisbetterthancomplex.com/tutorial/2018/11/28/advanced-form-rendering-with-django-crispy-forms.html

اضافه کردن فیلدهای جدید به user

1. اکیدا توصیه می شود که این مراحل را قبل از شروع هر پروژه ای انجام دهید.
2. نوشتن این کدها در مدل اپ اکانت

```
from django.db import models
from django.contrib.auth.models import AbstractUser
from django.utils import timezone
# Create your models here.
class User(AbstractUser):
    is_author = models.BooleanField(default=False, verbose_name="وضعیت نویسندگی")
    spacial_user = models.DateTimeField(
        default=timezone.now(), verbose_name="تاریخ انقضای اشتراک ویژه"
    )
    def is_spacial_user(self):
        if self.spacial_user > timezone.now():
            return True
        else:
            return False
    is_spacial_user.short_description = "کاربری ویژه"
```

3. اضافه کردن این متغییر به setting.py

```
AUTH_USER_MODEL = "account.User"
```

4. Admin.py اپ account

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from .models import User

admin.site.register(User, UserAdmin)
```

5. تغییر from django.contrib.auth.models import User به from account.models import User در مدل اپ blog
6. حذف کامل دیتابیس
7. Makemigration , migrate
8. py manage.py createsuperuser

اضافه کردن فیلدهای جدید کاربر به ادمین پنل

1. تغییر admin.py بصورت زیر

```
from django.contrib import admin
```

```
from django.contrib.auth.admin import UserAdmin
from .models import User

UserAdmin.fieldsets += (
    ("فیلد های اختصاصی", {"fields": ("is_author", "spacial_user")}),
)

UserAdmin.list_display += ("is_author", "is_spacial_user")
admin.site.register(User, UserAdmin)
```

2. اضافه کردن این خط در مدل

```
is_spacial_user.boolean = True
```

استفاده از mixin ها برای تعیین فیلدهای یک فرم

1. ایجاد یک فایل بنام mixins.py در اپ account و نوشتن کد های زیر

```
from django.http import Http404

class FieldMixin:
    def dispatch(self, request, *args, **kwargs):
        if request.user.is_superuser:
            self.fields = [
                "title",
                "slug",
                "category",
                "author",
                "thumbnail",
                "body",
                "status",
            ]
        elif request.user.is_author:
            self.fields = [
                "title",
                "slug",
                "category",
                "thumbnail",
                "body",
            ]
```

```
        else:
            raise Http404
        return super().dispatch(request, *args, **kwargs)

class FormValidMixin:
    def form_valid(self, form):
        if self.request.user.is_superuser:
            form.save()
        else:
            self.obj = form.save(commit = False)
            self.obj.author = self.request.user
            self.obj.status = 'draft'
        return super().form_valid(form)
```

2. تغییر ویوی ایجاد پست بصورت زیر

```
from .mixins import FieldMixin, FormValidMixin
class PostCreate(LoginRequiredMixin, FormValidMixin,FieldMixin, CreateView):
    model = Post
    template_name = "adminlte/post_create_update.html"
    success_url = reverse_lazy("account:home")
```

3. اصلاح بخشی از تمپلیت ایجاد پست بصورت زیر

```
<div class="form-row">
      {% if user.is_superuser %}
        <div class="form-group col-md-6 mb-0">
          {{ form.category|as_crispy_field }}
        </div>
        <div class="form-group col-md-6 mb-0">
          {{ form.author|as_crispy_field }}
          {{ form.status|as_crispy_field }}
        </div>

      {% else %}
      <div class="form-group col-md-12 mb-0">
        {{ form.category|as_crispy_field }}
      </div>
      {% endif %}
    </div>
    <div class="form-row">
```

```
      {{ form.thumbnail|as_crispy_field }}
      {{ form.body|as_crispy_field }}
    </div>
```

ویرایش مقالات در ادمین شخصی

1. ساخت ویوی آپدیت در اپ اکانت

```
from .mixins import FieldMixin, FormValidMixin, AuthorAccessMixin

class PostUpdate(AuthorAccessMixin, FormValidMixin, FieldMixin, UpdateView):
    model = Post
    template_name = "adminlte/post_create_update.html"
    success_url = reverse_lazy("account:home")
```

2. ساخت AuthorAccessMixin

```
from django.shortcuts import get_object_or_404
from blog.models import Post

class AuthorAccessMixin:
    def dispatch(self, request, pk, *args, **kwargs):
        post = get_object_or_404(Post, pk=pk)
        if post.author == request.user or request.user.is_superuser:
            return super().dispatch(request, *args, **kwargs)
        else:
            raise Http404('دسترسی غیر مجاز')
```

3. ساخت url آپدیت

```
path("post/update/<int:pk>", views.PostUpdate.as_view(), name="post_update")
```

4. اصلاح عنوان پست در home.html بخش ادمین شخصی

```
{% if obj.status == "draft" or user.is_superuser %}
<td>
  <a href = "{% url 'account:post_update' obj.pk %}">
    {{obj.title}}
  </a>
</td>
{% else %}
    <td>{{obj.title}}</td>
```

```
{% endif %}
```

حذف مقاله

1. ساخت یک mixin که فقط superuser بتواند حذف کند

```
class SuperuserAccessMixin:
    def dispatch(self, request, pk, *args, **kwargs):
        if request.user.is_superuser:
            return super().dispatch(request, *args, **kwargs)
        else:
            raise Http404("دسترسی غیر مجاز")
```

2. ساخت یک ویو برای حذف

```
class PostDelete(SuperuserAccessMixin, DeleteView):
    model = Post
    template_name = "adminlte/confirm_post_delete.html"
    success_url = reverse_lazy("account:home")
```

3. ساخت یک url برای حذف

```
path("post/delete/<int:pk>", views.PostDelete.as_view(), name="post_delete"),
```

4. ساخت تمپلیت تایید حذف

```
{% extends 'adminlte/base.html' %}
{% block title %}حذف مقالات{% endblock title %}
{% block main %}
<form method="post">{% csrf_token %}
    <p>آیا از حذف مقاله "{{ object }}" مطمئن هستید؟</p>
    {{ form }}
    <input type="submit" value="تایید">
</form>
{% endblock main %}
```

5. اصلاح ستون آخر جدول لیست مقالات در home.html

```
{% if user.is_superuser %}
   <div >
       <a href = {% url 'account:post_delete' obj.pk %} type="button"
class="label pull-left bg-red">حذف</a>
```

```
   </div>
{% endif %}
```

Log out در ادمین شخصی

1. ساخت لوگو برای لاگ اوت_ قراردادن تگ های زیر در صفحه base.html

```
<div class="pull-left">
 <a href="{% url 'account:logout' %}" class="btn btn-default btn-flat">خروج</a>
</div>
```

2. اصلاح logout url

```
path("logout/", base_view.LogoutView.as_view(), name="logout"),
```

3. اضافه کردن این متغییر در setting.py

```
LOGOUT_REDIRECT_URL = "account:login"
```

اضافه کردن کلاس active به لینک های فعال در سایدبار

1. ایجاد یک template tag در base_tag.py

```
@register.inclusion_tag("adminlte/partials/active_link.html")
def link(request, link_name, content, class_):
    return {
        "request": request,
        "link": f"account:{link_name}",
        "link_name": link_name,
        "content": content,
        "class": class_,
    }
```

```
    }
```

2. ایجاد صفحه adminlte/partials/active_link.html

```
<li
    {% if request.resolver_match.url_name == link_name %}
        class="active"
    {% endif %}
>
```

```
    <a href="{% url link %}">
        <i class="fa {{class}}"></i>
        {{content}}
    </a>
</li>
```

3. ایجاد تغییر در sidebar.html و قرار دادن تگ {٪ load base_tag ٪} در ابتدای صفحه

```
          <ul class="treeview-menu">
              {% link request 'home' "لیست مقالات" 'fa-list' %}
              {% link request 'post_create' 'ایجاد مقاله' 'fa-edit' %}

          </ul>
```

4. Restart کردن سرور (همیشه یادمان باشد برای رجیستر کردن tamplate tag های جدید حتما باید سرور restart شود)

قرار دادن دکمه حذف مقاله فقط برای superuser در پایین صفحه ویرایش مقاله

1. قراردادن این کد در صفحه ایجاد یا ویرایش مقاله

```
{% if user.is_superuser and request.resolver_match.kwargs.pk %}
  <a href = "{% url 'account:post_delete' request.resolver_match.kwargs.pk %}"
    class = "btn btn-danger">
    حذف
  </a>
{% endif %}
```

اضافه کردن قابلیت پیش نمایش به مقالات در ادمین پنل شخصی

1. ایجاد ویو در views.py اپ بلاگ

```
from account.mixins import AuthorAccessMixin
class PostPreview(AuthorAccessMixin, DetailView):
    def get_object(self):
        pk = self.kwargs.get("pk")
        return get_object_or_404(Post, pk=pk, status="draft")

    context_object_name = "post"
```

```
    template_name = "blog/post/detail.html"
```

2. ایجاد url در url.py اپ بلاگ

```
path("preview/<int:pk>", views.PostPreview.as_view(), name="post_preview"),
```

3. ساخت لینک پیشنمایش مقاله در صفحه ادمین شخصی در home.html

```
<a href = {% url 'blog:post_preview' obj.pk %} type="button" class="btn btn-block btn-scondary">پیشنمایش</a>
```

افزایش وضعیت مقالات

1. اصلاح مدل پست

```
    STATUS_CHOICES = (
        ("draft", "پیشویس"),
```

```
        ("published", "منتشر شده"),
        ("peding", "در دست بررسی"),
        ("back", "برگشت داده شده"),
    )
```

2. اصلاح mixin

```
class AuthorAccessMixin:
    def dispatch(self, request, pk, *args, **kwargs):
        post = get_object_or_404(Post, pk=pk)
        if post.author == request.user and post.status in ['draft','back'] or request.user.is_superuser:
            return super().dispatch(request, *args, **kwargs)
        else:
            raise Http404("دسترسی غیر مجاز")
```

3. اصلاح تمپلیت home.html

```
{% if obj.status == "draft" or obj.status == "back"  or user.is_superuser %}
              <td>
                <a href = "{% url 'account:post_update' obj.pk %}">
                  {{obj.title}}
                </a>
```

```
            </td>
{% else %}
                <td>{{obj.title}}</td>
{% endif %}
```

```
{% if obj.status == 'published' %}
  <div>
      <span class="label label-success">منتشر شده</span>
  </div>
  <div>
      <a href = {% url 'blog:detail_post' obj.slug %} type="button" class="btn
btn-block btn-primary">مشاهده مقاله</a>
  </div>
  {% elif obj.status == 'peding' %}
  <div>
      <span class="bg-yellow-active color-palette">در دست بررسی</span>
  </div>
    {% if user.is_superuser %}
      <div>
        <a href = {% url 'blog:detail_post' obj.slug %} type="button" class="btn
btn-block btn-primary">مشاهده مقاله</a>
      </div>
    {% endif %}
  {% elif obj.status == 'back' %}
    <div>
        <span class="bg-gray-active color-palette">برگشت خورده</span>
    </div>
  {%else%}
      <div class="bg-yellow color-palette">
          <span>پیشنویس</span>
      </div>
      <a href = {% url 'blog:post_preview' obj.pk %} type="button" class="btn
btn-block btn-scondary">پیشنمایش</a>
  {% endif %}
```

مقالات ویژه در ازای اشتراک ویژه

1. اضافه کردن این فیلد به مدل پست

```
is_special = models.BooleanField(default=False, verbose_name="مقاله ویژه")
```

2. Makemigrations , migrate
3. اضافه کردن فیلد جدید به admin.py

```
list_display =
("title","slug","author","is_special","jalali_publish","status","all_category",)
```

4. اضافه کردن فیلد جدید در fieldmixin اپ account

```
class FieldMixin:
    def dispatch(self, request, *args, **kwargs):
        if request.user.is_superuser:
            self.fields =
["title","slug","category","author","thumbnail","is_special","body","status",]
        elif request.user.is_author:
            self.fields =
["title","slug","category","is_special","thumbnail","body",]
        else:
            raise Http404
        return super().dispatch(request, *args, **kwargs)
```

5. اضافه کردن فیلد جدید به تمپلیت اضافه کردن یا ویرایش مقاله

```
{{ form.is_special|as_crispy_field }}
```

6. اضافه کردن این کد های در تمپلیت detail.html

```
{% if post.is_special %}
    {% if user.is_authenticated and user.is_special or post.author == user or
user.is_superuser %}
        <p>{{post.body|linebreaks}}</p>
    {% else %}
        <p> برای مشاهده این مقاله روی <a href = '#'>خرید اشتراک</a>  کلیک کنید.</p>
    {% endif %}
{% else %}
<p>{{post.body|linebreaks}}</p>
{% endif %}
```

7. اضافه کردن یک ستون برای نمایش وضعیت مقالات ویژه در صفحه ادمین پنل شخصی

```
<td>
                    {% if obj.is_special %}
                    <i class="fa fa-check-circle" style = 'color: green'></i>
```

```
                {% else %}
                <i class="fa fa-minus-circle" style = 'color: red'></i>
                {% endif %}
    </td>
```

ساخت پروفایل کاربری

1. ساخت view

```
class Profile (UpdateView):
    model = User
    template_name = "adminlte/profile.html"
    fields = ['username','email',
'first_name','last_name','spacial_user','is_author']
    success_url = reverse_lazy("account:profile")

    def get_object(self):
        return User.objects.get(pk = self.request.user.pk)
```

2. ساخت url

```
path("user/profile", views.Profile.as_view(), name="profile"),
```

3. ساخت تمپلیت profile.html

```
{% extends 'adminlte/base.html' %}
{% load crispy_forms_tags %}

{% block title %}پروفایل{% endblock title %}
{% block main %}
 <form method="post" enctype="multipart/form-data" >
    {% csrf_token %}
    <div class="form-row">
      <div class="form-group col-md-6 mb-0">
        {{ form.username|as_crispy_field }}
      </div>
      <div class="form-group col-md-6 mb-0">
        {{ form.first_name|as_crispy_field }}
        {{ form.last_name|as_crispy_field }}
      </div>
    </div>

    <div class="form-row">
```

```
      {{ form.is_author|as_crispy_field }}
      {{ form.spacial_user|as_crispy_field }}
    </div>
    <button type = 'submit' class="btn btn-primary">ویرایش</button>

 </form>
{% endblock main %}
```

تغییر ویژگی های صفحه پروفایل با استفاده از فرم ها در جنگو

1. اضافه کردن یک فایل جدید به اپ اکانت با نام forms.py و نوشتن کد های زیر در آن

```
from django import forms
from .models import User

class ProfileForm(forms.ModelForm):
    def __init__(self, *args, **kwargs):
        user = kwargs.pop("user")
        super().__init__(*args, **kwargs)
        if not user.is_superuser:
            self.fields["username"].disabled = True
            self.fields["email"].disabled = True
            self.fields["spacial_user"].disabled = True
            self.fields["is_author"].disabled = True

    class Meta:
        model = User
        fields = [
            "username",
            "email",
            "first_name",
            "last_name",
            "spacial_user",
            "is_author",
        ]
```

2. تغییر ویوی profile به شکل زیر

```
from .forms import ProfileForm
```

```
class Profile(LoginRequiredMixin,UpdateView):
    model = User
    template_name = "adminlte/profile.html"
    form_class = ProfileForm
    success_url = reverse_lazy("account:profile")

    def get_object(self):
        return User.objects.get(pk=self.request.user.pk)

    def get_form_kwargs(self):
        kwargs = super().get_form_kwargs()
        kwargs["user"] = self.request.user
        return kwargs
```

3. ایجاد لینک پروفایل در سایدبار

```
<li>
    {% link request 'profile' 'پروفایل' 'fa fa-th' %}
</li>
```

4. اضافه کردن این کلاس در views.py اپ account جهت ریدایرکت بعد از لاگین شدن

```
from django.contrib.auth.views import LoginView

class Login(LoginView):
    def get_success_url(self):
        user = self.request.user
        if user.is_superuser or user.is_author:
            return reverse_lazy('account:home')
        else:
            return reverse_lazy('account:profile')
```

5. اصلاح مسیر در urls.py

```
path("login/", views.Login.as_view(), name="login"),
```

6. غیر فعال کرن برخی گزینه ها برای کاربران عادی در سایدبار با استفاده از دستورات زیر

```
{% if use.is_superuser or user.is_author %}
  <li class="active treeview">
    <a href="#">
```

```
      <i class="fa fa-dashboard"></i> <span>مقالات</span>
      <span class="pull-left-container">
        <i class="fa fa-angle-right pull-left"></i>
      </span>
    </a>
    <ul class="treeview-menu">
        {% link request 'home' "لیست مقالات" 'fa-list' %}
        {% link request 'post_create' 'ایجاد مقاله' 'fa-edit' %}
    </ul>
  </li>
{% endif %}
```

7. اضافه کردن کلاس زیر در mixin

```
from django.shortcuts import get_object_or_404, redirect

class AuthorsAccessMixin:
    def dispatch(self, request, *args, **kwargs):
        if request.user.is_superuser or request.user.is_author:
            return super().dispatch(request, *args, **kwargs)
        else:
            raise redirect("account:profile")
```

8. تغییر mixin کلاس های PostUpdateو PostCreate از LoginRequiredMixin به AuthorsAccessMixin

تغییر گذرواژه در ادمین پنل شخصی

1. ساخت url

```
path("password_change/", views.PasswordChange.as_view(),
name="password_change"),
```

2. اضافه کردن این کلاس در views.py

```
from django.contrib.auth.views import LoginView, PasswordChangeView

class PasswordChange(PasswordChangeView):
    template_name = "adminlte/password_change_form.html"
    success_url = reverse_lazy("account:profile")
```

3. ایجاد یک تمپلیت با نام password_change_form.html در adminlte

```
{% extends 'adminlte/base.html' %}
{% load crispy_forms_tags %}

{% block title %}تغییر گذرواژه{% endblock title %}
{% block main %}
 <form method="post" >
    {% csrf_token %}
    <div class="form-row">
      <div class="form-group col-md-6 mb-0">
        {{ form.old_password|as_crispy_field }}
      </div>
      <div class="form-group col-md-6 mb-0">

      </div>
    </div>
    <div class="form-row">
        <div class="form-group col-md-5 mb-0">
            {{ form.new_password1|as_crispy_field }}
          </div>
      <div class="form-group col-md-5 mb-0">
        {{ form.new_password2|as_crispy_field }}
      </div>

    </div>
    <div class="form-row">
        <button type = 'submit' class="btn btn-primary">تغییر گذرواژه</button>
    </div>
 </form>
{% endblock main %}
```

4. ایجاد لینک های تغییر گذرواژه در سایدبار

اصلاح شیوه لاگین

1. اضافه کردن مسی لاگین در urls.py اصلی config

```
from account.views import Login
path("login/", Login.as_view(), name="login"),
```

2. اصلاح متغییر های url های مربوط به لاگین در setting.py

```
LOGIN_URL = "login"
```

```
LOGOUT_REDIRECT_URL = "login"
```

ارسال لینک بازیابی گذرواژه به ایمیل بصورت کنسولی

1. رفتن به مسیر http://127.0.0.1:8000/password_reset/ و نمایش شیوه ریست کردن گذر واژه
2. اضافه کردن متغییر زیر در صفحه setting.py

```
EMAIL_BACKEND = "django.core.mail.backends.console.EmailBackend"
```

3. تست کردن ارسال ایمیل با نوشتن این کد در ترمینال

```
Py manage.py runserver
```

4. اضافه کردن این لینک به تمپلیت لاگین

```
<a href = "{% url 'password_reset' %}">فراموشی رمز عبور</a>
```

5. با رفتن به آدرس لینک بالا و نوشتن آدرس ایمیل درصورت ثبت شدن آدرس ایمیل در پایگاه داده ایمیلی به کاربر ارسال خواهد شد. با توجه به اینکه در اینجا از console backend email استفاده کردیم ایمیل ارسال نشده و فقط در کنسول نمایش داده می شود.
6. با توجه به اینکه ایمیل بطور پیشفرض در جنگو اجباری و یکتا تعریف نشده است لازم است که این فیلد را در moles.py اپ account اضافه کنیم.

```
email = models.EmailField(unique=True, verbose_name='ایمیل')
```

7. Migrate, makemigrations

بومی سازی صفحات مربوط به بازیابی و یا تغییر گذر واژه

ارسال کردن جیمل در جنگو

1. جستجور کردن send gmail in Django
2. اقدام براساس راهنمایی های سایت https://bshoo.medium.com/how-to-send-emails-with-python-django-through-google-smtp-server-for-free-22ea6ea0fb8e
3. کپی کردن و قراردادن کدهای زیر در setting.py

```
EMAIL_BACKEND = "django.core.mail.backends.smtp.EmailBackend"
EMAIL_HOST = "smtp.gmail.com"
EMAIL_USE_TLS = True
EMAIL_PORT = 587
EMAIL_HOST_USER = "your_account@gmail.com"
EMAIL_HOST_PASSWORD = "your account's password"
```

ثبت نام در سایت

1. جستجوی واژه django signup user with email confirmation در گوگل
2. مراجعه به این سایت و عمل براساس راهنمایی های آن

https://medium.com/@frfahim/django-registration-with-confirmation-email-bb5da011e4ef

3. ساخت یک فایل tokens.py در اپ account و کپی کدهای زیر در آن

```
from django.contrib.auth.tokens import PasswordResetTokenGenerator
from django.utils import six
class TokenGenerator(PasswordResetTokenGenerator):
    def _make_hash_value(self, user, timestamp):
        return (
            six.text_type(user.pk) + six.text_type(timestamp) +
            six.text_type(user.is_active)
        )
account_activation_token = TokenGenerator()
```

4. کپی این کد در forms.py

```
from django.contrib.auth.forms import UserCreationForm

class SignupForm(UserCreationForm):
    email = forms.EmailField(max_length=200, help_text='Required')
    class Meta:
        model = User
        fields = ('username', 'email', 'password1', 'password2')
```

5. کپی کردن این کدها در views.py

```
from django.http import HttpResponse
from django.shortcuts import render, redirect
from django.contrib.auth import login, authenticate
from .forms import SignupForm
from django.contrib.sites.shortcuts import get_current_site
from django.utils.encoding import force_bytes, force_str
from django.utils.http import urlsafe_base64_encode, urlsafe_base64_decode
from django.template.loader import render_to_string
from .tokens import account_activation_token
from django.core.mail import EmailMessage
def signup(request):
    if request.method == 'POST':
        form = SignupForm(request.POST)
        if form.is_valid():
```

```
            user = form.save(commit=False)
            user.is_active = False
            user.save()
            current_site = get_current_site(request)
            mail_subject = 'Activate your blog account.'
            message = render_to_string('adminlte/acc_active_email.html', {
                'user': user,
                'domain': current_site.domain,
                'uid':urlsafe_base64_encode(force_bytes(user.pk)),
                'token':account_activation_token.make_token(user),
            })
            to_email = form.cleaned_data.get('email')
            email = EmailMessage(
                        mail_subject, message, to=[to_email]
            )
            email.send()
            return HttpResponse('Please confirm your email address to complete
the registration')
    else:
        form = SignupForm()
    return render(request, 'adminlte/signup.html', {'form': form})
```

6. ساخت تمپلیت `acc_active_email.html`

```
{% autoescape off %}
    Hi {{ user.username }},
    Please click on the link to confirm your registration,
    http://{{ domain }}{% url 'activate' uidb64=uid token=token %}
{% endautoescape %}
```

7. ساخت یک ویوی جدید

```
def activate(request, uidb64, token):
    try:
        uid = force_str(urlsafe_base64_decode(uidb64))
        user = User.objects.get(pk=uid)
    except (TypeError, ValueError, OverflowError, User.DoesNotExist):
        user = None
    if user is not None and account_activation_token.check_token(user, token):
        user.is_active = True
        user.save()
        login(request, user)
```

```
        # return redirect('home')
        return HttpResponse(
            "Thank you for your email confirmation. Now you can login your account."
        )
    else:
        return HttpResponse("Activation link is invalid!")
```

8. اضافه کردن url ها

```
path(r"^signup/$", views.signup, name="signup"),
    re_path(
        r"^activate/(?P<uidb64>[0-9A-Za-z_\-]+)/(?P<token>[0-9A-Za-z]{1,13}-[0-9A-Za-z]{1,20})/$",
        views.activate,
        name="activate",
    ),
```

9. نصب pip install django-utils-six

10. اضافه کردن تمپلیت `signup.html`

```
{% extends 'adminlte/base.html' %}
{% block main %}
  <h2>Sign up</h2>
  <form method="post">
    {% csrf_token %}
      {% for field in form %}
      <p>
        {{ field.label_tag }}<br>
        {{ field }}
        {% if field.help_text %}
          <small style="display: none">{{ field.help_text }}</small>
        {% endif %}
        {% for error in field.errors %}
          <p style="color: red">{{ error }}</p>
        {% endfor %}
      </p>
      {% endfor %}
    <button type="submit">Sign up</button>
  </form>
{% endblock %}
```

دیپلوی

https://www.mongard.ir/one_part/65/deploying-django-projects/

https://www.fandogh.cloud/languages/python

https://www.codingyar.com/posts/%D8%AF%DB%8C%D9%BE%D9%84%D9%88%DB%8C-%D8%AC%D9%86%DA%AF%D9%88-%D8%AF%D8%B1-%D9%84%DB%8C%D8%A7%D8%B1%D8%A7/

Enhancing Your Blog with Advanced Features

Create forms.py in blog

```
from django.forms.widgets import Textarea

class EmailPostForm(forms.Form):
    name=forms.CharField(max_length=25)
    email=forms.EmailField()
    to=forms.EmailField()
    comments=forms.CharField(required=False,widget=forms.Textarea)
```

views.py

```
from .forms import EmailPostForm
def post_share(request,post_id):
    # Retrieve post by id
    post=get_object_or_404(Post,id=post_id,status='published')
    if request.method == 'POST':
        # Form was submitted
        form=EmailPostForm(request.POST)
        if form.is_valid():
            # Form fields passed validation
            cd=form.cleaned_data
            # send email
    else:
        form=EmailPostForm()
```